

OFFICE EUROPÉEN DE LUTTE ANTI-FRAUDE

ANTI-FRAUD INFORMATION SYSTEM (AFIS)

B2B

Technical Specifications

DOCUMENT HISTORY

Ver.	Release Date	Description	Author (Function, Party)
1.0	23/04/2021	Initial version	Dario Savarese Pedro do Vale
1.1	16/06/2021	• 1.2.3.1.2 more details; how to compute cnf_key ; how to generate client_assertion kid • 1.2.4.1.1 details on how to compute the DPoP ath claim ath	Dario Savarese Pedro do Vale
1.2	09/02/2022	• Entire document: Changed the wording from **DPoP token** to **DPoP proof** (to avoid confusion with the access token of type DPoP) • 1.2.3.1.1 added attention point for DPoP header validation • 1.2.3.1.2 updated details for multiple scopes • 1.2.3.1.2 client_id removed from the body parameters (not needed since it is contained in the client_assartion claims) • 1.2.3.1.2 more details: how to generate client_assertion kid • 1.2.3.1.2 more details: exp claim in the client_assertion • Headers1.2.4.1.1 more details about the 2 headers specific to the B2B protocol • 1.3 Demonstrator clarifications • 2 SAFE Demo Client clarifications and more detailed explanation of the endpoints Explanations adapted to version 0.3.0 of the SAFE Demo Client • Protocol implementation: SAFE2.4 Protocol implementation: SAFE • Explanations adapted to version 0.3.0 of the SAFE library • Removed section “Verifying SAFE requests/responses using Wireshark”	Dario Savarese Pedro do Vale
3.0	08/06/2026	• Align version number with API Spec version 3.0	Frank Crijns

		• Mention that DPoP will be accepted, but not validated any more • Remove SAFE Client (unsupported)	

REFERENCE DOCUMENTS

Title	Reference	Version	Date
The OAuth 2.0 Authorization Framework	https://tools.ietf.org/html/rfc6749	-	10/2012
OAuth 2.0 Security Best Current Practice (SBCP)	https://tools.ietf.org/html/draft-ietf-oauth-security-topics-16	16	05/10/2021
OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP)	https://tools.ietf.org/html/draft-ietf-oauth-dpop-03	3	07/04/2021
OAuth2.0 DPoP draft specification tracker	https://datatracker.ietf.org/doc/html/draft-ietf-oauth-dpop-03	3	-
OpenID Connect Core	https://openid.net/specs/openid-connect-core-1_0.html	1	8/11/2014
JSON Web Token (JWT)	https://tools.ietf.org/html/rfc7519	-	05/2015
JSON Web Key (JWK)	https://tools.ietf.org/html/rfc7517	-	05/2015
JSON Web Signature (JWS)	https://tools.ietf.org/html/rfc7515	-	05/2015
JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants	https://tools.ietf.org/html/rfc7523	-	05/2015
Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)	https://tools.ietf.org/html/rfc7800	-	04/2016
Resource Indicators for OAuth 2.0	https://tools.ietf.org/html/rfc8707	-	02/2020
OAuth 2.0 Token Introspection	https://tools.ietf.org/html/rfc7662	-	10/2015
The 'Basic' HTTP Authentication Scheme	https://tools.ietf.org/html/rfc7617	-	09/2015
Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing	https://datatracker.ietf.org/doc/html/rfc7230	-	06/2014
Spring Security Framework OAuth2 capabilities	https://docs.spring.io/spring-security/site/docs/current/reference/html5/#oauth2	5.4.5	17/02/2021
Spring Boot	https://spring.io/projects/spring-boot	2.4.4	2021
Forgerock Access Management Documentation	https://backstage.forgerock.com/docs/am/7	7	24/03/2021
Java Platform, Standard Edition (Java SE) 8	JDK 8 and JRE 8 Installation	-	-
AFIS B2B technical specifications without DPoP	AFIS B2B technical specifications without DPoP v3.0.pdf	3.0	10/06/2026

GLOSSARY

AFIS	**Anti-Fraud Information System**
alg	**(JWT) Algorithm**
AM	**Forgerock Access Management**
AS	**OAuth2.0 Authorization Server**
AT	**Access Token**
ath	**Access Token Hash**
aud	**JWT Audience**
B2B	**Business-to-Business**
cnf	**confirmation**
CS	**Client System**
DPoP	**Demonstrating Proof-of-Possession**
exp	**JWT Expiration time**
htm	**DPoP Proof HTTP Method**
HTTP	**Hypertext Transfer Protocol**
htu	**DPoP Proof HTTP URI**
iat	**JWT Issued At time**
iss	**JWT Issuer**
JSON	**Javascript Object Notation**
jti	**JWT Identifier**
JWK	**JSON Web Key**
JWKS	**JSON Web Key Set**
JWS	**JSON Web Signature**
JWT	**JSON Web Token**
OAuth2.0	**Open Authorization 2.0**
PoP	**Proof-of-Possession**
RS	**OAuth2.0 Resource Server**
RT	**Refresh Token**
SBCP	**OAuth2.0 Security Best Current Practices**
sub	**JWT Subject**
typ	**JWT Type**
URI	**Uniform Resource Identifier**
UUID	**Universal Unique Identifier**

Contents

DOCUMENT HISTORY 2

REFERENCE DOCUMENTS 4

GLOSSARY 5

1 Accessing AFIS B2B services 7

1.1 Introduction 7

1.2 Protocol 7

1.2.1 Prerequisites 8

1.2.2 Steps 8

1.2.3 Authorization Server Exchange 9

1.2.4 Resource Server Exchange 14

2 Annexes 17

2.1 AFIS Security Best Practice for B2B 17

2.1.1 Abstract 17

2.1.2 Recommendations 17

2.2 AFIS DPoP 19

2.2.1 Introduction 19

2.2.2 Main Objective 20

2.2.3 Concept 20

2.2.4 DPoP Proof JWT Syntax 21

2.2.5 Token Request (Binding Tokens to a Public Key) 23

2.2.6 Public Key Confirmation 24

2.2.7 AFIS DPoP Flow Implementation 26

1 Accessing AFIS B2B services

1.1 Introduction

An AFIS B2B service is a service meant to be consumed by an application rather than by a human user. In order to consume an AFIS B2B service, the calling application must be trusted and entitled. This involves authentication (i.e. determining who the caller is) and authorization (i.e. assessing what the caller can do).

In order to provide an adequate level of security, the process relies on the OAuth 2.0 protocol using the Client Credentials grant type with the following additional practices:

- authentication is performed by means of the private_key_jwt method as specified in OpenID Connect Core 1.0;
- the Access Token is Sender-Constrained by means of DPoP (Demonstrating Proof-of-Possession at the Application Layer) as specified in DPoP IETF Draft Specification;
- the Access Token is Audience-Restricted by means of Resource Indicators, as specified in RFC 8707.

⚠ ***** Attention *****
The latest version of the AFIS B2B services still accepts DPoP proof tokens, but they will not be actively validated any more. It is strongly advised to switch to OAuth2 authorization without DPoP.

This document is provided for legacy support. The most up-to-date technical specifications can be found in *AFIS B2B technical specifications with DPoP v3.0.pdf*

This document details the protocol that must be used for accessing AFIS B2B services and provides guidance to use the demonstrator implementing the protocol and distributed under the European Union Public License (EUPL).

1.2 Protocol

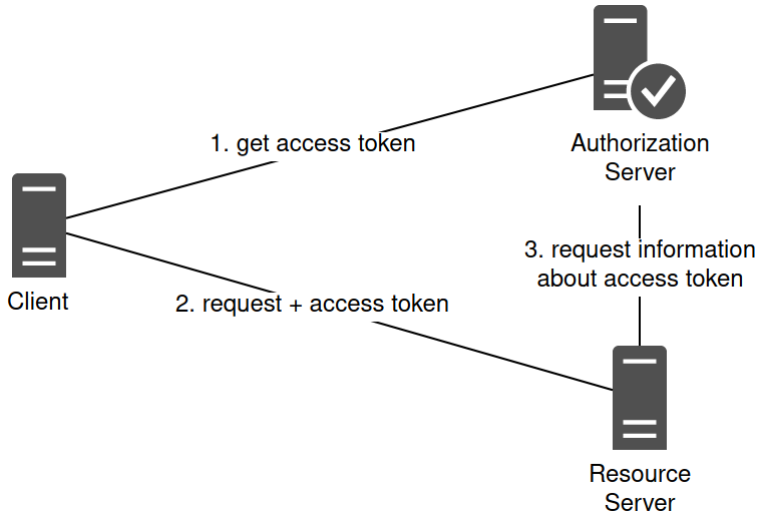
OAuth2.0 is an open standard that addresses authorization and, to some extent, authentication. It supports several usage scenarios for which specific flows are specified, known as grant types.

The Client Credentials grant type specifically targets the use case where an application invokes another application. The calling application, known as the **Client**, obtains an **Access Token** from a trusted third-party, known as the **Authorization Server**, in order to consume a service exposed by a B2B application, known as the **Resource Server**. This requires that the Client trusts the Authorization Server.

The **Resource Server** can obtain information on the **Client** in order to perform access control by providing the **Access Token** to the **Authorization Server**. This requires that the Resource Server trusts the Authorization Server.

The Authorization Server therefore acts as a trusted third-party allowing the Resource Server to establish trust with the Client.

The figure below provides a high-level view of the OAuth2.0 Client Credentials grant type.



1.2.1 Prerequisites

The prerequisites to implement the protocol are:

- An RSA public/private key pair with a 2048-bit modulus in order to authenticate the caller - This is further referred to as **AuthN Key Pair**.
- Registration of the public key of the AuthN Key Pair at AFIS level – As a result of this process, a client identifier is provided (e.g. example-client), further referred to as **client_id**. Optionally, the public key can be dynamic and served as a JSON Web Key Set (JWKS) from a URL.
- An RSA public/private key pair with a 2048-bit modulus in order to implement DPoP. It can be a permanent key pair, possibly identical to the AuthN Key Pair, or, preferably, a transient key pair generated on the spot. This is further referred to as **DpoP Key Pair**. The public key does not need to be registered at AFIS level.
- Knowledge of the resource indicator of the Authorization Server to be invoked. It corresponds to the absolute URI of the AFIS Authorization Server. This is further referred to as **Authorization Audience**.
- Knowledge of the resource indicator of the Resource Server to be invoked. It corresponds to the absolute URI of the AFIS B2B Service called by the Client and is provided as part of the definition of the AFIS B2B Service (e.g. https://resource-server.sample/protected-resource). This is further referred to as **Resource Audience**.
- Knowledge of the scopes to be bound to the Access Token. They are used to limit what the Access Token grants access to. It is defined for each Resource Server and it can be obtained as part of the definition of the AFIS B2B Service. This is further referred to as **Resource Server Scopes**.

1.2.2 Steps

The protocol happens in two steps:

1. obtaining an access token from the Authorization Server — this is further referred to as the **Authorization Server Exchange**;
2. using the access token to invoke the Resource Server — this is further referred to as the **Resource Server Exchange**.

1.2.3 Authorization Server Exchange

The Authorization Server Exchange is initiated by the Client that sends a request to the Authorization Server. It allows the Authorization Server to authenticate the client.

The Authorization Server answers with a response containing an Access Token that the client can use in order to invoke an AFIS B2B Service.

1.2.3.1 Request

The figure below describes the request sent by the Client using HTTPS.

```
POST /token HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Dpop: eyJraWQiOiJkNTA…
Host: authorization-server.sample

grant_type=client_credentials
&scope=read
&client_id=example-client
&cnf_key=eyJqd2siOiB7Imt...
&resource=https%3A%2F%2Fresource-server.sample%2Fprotected-resource
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-
type%3Ajwt-bearer
&client_assertion=eyJraWQiOiI3NDQ...
```

The first line of the request contains the method that must be set to `POST`. It is followed by the headers above the blank line and the body under the blank line.

1.2.3.1.1 Headers

In addition to the usual HTTP headers, the request must include a DPoP header containing a JSON Web Token encoded using Base64 further referred to as Authorization Server DPoP Proof.

The Authorization Server DPoP Proof must be signed with the private key of the DPoP Key Pair (see Prerequisites).

The figure below describes the Authorization Server DPoP Proof.

```
{
  "typ": "dpop+jwt",
  "alg": "RS256",
  "jwk": {
    "kty": "RSA",
    "e": "AQAB",
    "use": "sig",
```

```
    "kid": "a34d9c68-7170-42d0-8792-7531c8a28237",
    "n": "lzweTKEPtiol--U..."
  }
}
{
  "htu": "https://authorization-server.sample/token",
  "htm": "POST",
  "iat": 1615987167,
  "jti": "ff6b3fc6-6a7e-4cf7-8cbe-4a320e5a844b"
}
```

The Authorization Server DPoP Proof is different from the JSON Web Token (JWT) used to authenticate the Client.

The first part is a Javascript Object Signing and Encryption (JOSE) header with the following elements:

- `typ` – The media type of the complete JWT. For accessing AFIS B2B Services, the value must always be `dpop+jwt`.
- `alg` – The digital signature algorithm. For accessing AFIS B2B Services, the value must always be `RS256`. It corresponds to signing using a SHA-256 hash and RSA encryption.
- `jwk` – The public key of the DPoP Key Pair represented using the JSON Web Key (JWK) format.

The second part, commonly referred to as claims, has the following elements:

- `htu` – The URI of the Authorization Server token endpoint.
- `htm` – The HTTP method used to invoke the Authorization Server token endpoint. For accessing AFIS B2B Services, the value must always be `POST`.
- `iat` – The time when the JWT was issued, expressed as an Epoch time (i.e. number of seconds elapsed since Thursday January 1st 1970 at 0:00:00 GMT/UTC).When received by the Authorization Server, the DPoP proofshould not be older than a few minutes.
- `jti` – The unique identifier of the JWT. It is recommended to use a randomly non-sequential value, such as a Universally Unique Identifier (UUID).

The signature of the JWT is a JSON Web Signature (JWS), as defined in RFC 7515, performed using the private key of the DPoP Key Pair.

⚠ ***** Attention *****
The latest version of the AFIS B2B services still accepts DPoP proof tokens, but they will not be actively validated any more. It is strongly advised to switch to OAuth2 authorization without DPoP.

This document is provided for legacy support. The most up-to-date technical specifications can be found in *AFIS B2B technical specifications with DPoP v3.0.pdf*

1.2.3.1.2 Body

The following parameters are mandatory and must be encoded using percent-encoding, also known as URL encoding (e.g. `http://foo` becomes `http%3A%2F%2Ffoo`):

- **`grant_type`** – The OAuth 2.0 grant type used to obtain the Access Token. For accessing AFIS B2B Services, the value must always be `client_credentials`.
- **`scope`** – The Resource Server Scopes to which the Access Token is bound (see Prerequisites). If it is multi-valued then the values must be space-delimited, e.g. `'read write'`.
- **`client_id`** – The client_id obtained when registering the public key of the AuthN Key Pair (see Prerequisites).
- **`cnf_key`** – The public key of the DPoP Key Pair (see Prerequisites) represented as JWK, then encoded using Base64. For details see the figure below.
- **`resource`** – The Resource Audience corresponding to the AFIS B2B Service that will be invoked (see Prerequisites).
- **`client_assertion_type`** – The type of the client assertion. For accessing AFIS B2B Services, the value must always be `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`.
- **`client_assertion`** – The JSON Web Token (JWT) used to authenticate the Client, encoded using Base64 (see below). It must be signed with the private key of the AuthN Key Pair (see Prerequisites).

The figure below describes how to compute the cnf_key parameter:

Step 1. Create a JSON Object named "jwk" which value is the JWK representation of the Public Key of the DPoP Key Pair. For example:

```
{
  "jwk": {
    "kty": "RSA",
    "e": "AQAB",
    "use": "sig",
    "kid": "57c986c9-83de-343f-9f9c-c519e1da5938",
    "n": "kuMJw-
M4j83OVj1MygGUsekOlKGbgZQGHMxTVw_G1GEsNv4E4QFS6DbvLJQpdeMUTuOPniLtCBTDk9fLF
jbuCZzgWOPTW5T1EqiF59R_K3oEYc6Q-
v_Tjl7MYiDWhPD0hdn6PK_DxOMcYQYeLSDO_tEGZhV7aCKQ2eNTbc_kZtk6fFotHClzGbEhWD6A
U5BpVRDV0Pgur3Jip9taaAta4UmKi9-
ozfuKervUcYqghLebIo83AJd_y2APjeYg23rQPtdmehfi5Gi2gDCx6Iwizgik3Bj8VTn3klIHYH
TwfCwa70cIlZjTjbcqg6Za6R55aFt-t-PpI_d-IU7KoUiOJw"
  }
}
```

Step 2. Base64-encode the jwk json object. For example:

```
ewogICJqd2siOiB7CiAgICAia3R5IjogIlJTQSIsCiAgICAiZSI6ICJBUUFCIiwKICAgICJ1c2U
iOiAic2lnIiwKICAgICJraWQiOiAiNTdjOTg2YzktODNkZS0zNDNmLTlmOWMtYzUxOWUxZGE1OT
M4IiwKICAgICJuIjogImt1TUp3LU0ajgzT1ZqMU15Z0dVc2VrT2xLR2JnWlFHSE14VFZ3X0cxR
0VzTnY0RTRRRlM2RGJ2TEpRcGRlTVVUdU9QbmlMdENCVERrOWZMRmpidUNaemdXT1BUVzVUMUVx
aUY1OVJfSzNvRVljNlEtdl9Uamw3TVlpRFdoUEQwaGRuNlBLX0R4T01jWVFZZUxTRE9fdEVHWmh
WN2FDS1EyZU5UYmNfa1p0azZmRm90SENsekdiRWhXRDZBVTVCcFZSRFYwUGd1cjNKaXA5dGFhQX
RhNFVtS2k5LW96ZnVLZXJ2VWNZcWdoTGViSW84M0FKZF95MkFQamVZZzIzclFQdGRtZWhmaTVHa
TJnREN4Nkl3aXpnaWszQmo4VlRuM2tsSUhZSFR3ZkN3YTcwY0lsWmpUamJjcWc2WmE2UjU1YUZ0
LXQtUHBJX2QtSVU3S29VaU9KdyIKICB9Cn0=
```

The public key of the DPoP key pair will later be used to validate the Resource Server Exchange

The figure below describes the JWT used to authenticate the Client (client_assertion):

```
{
  "kid": "c8f20dcf-fcd4-4200-b32a-bcf8721cf68d",
  "typ": "jwt",
  "alg": "RS256"
}
{
  "aud": "https://authorization-server.sample/token",
  "sub": "example-client",
  "iss": "example-client",
  "exp": 1613561484,
  "iat": 1613560584,
  "jti": "b50fdb74-1bc0-4aff-9a0e-aa0b40dbaebe"
}
```

The JSON Web Token (JWT) used to authenticate the Client is different from the Authorization Server DPoP Proof

The first part is a Javascript Object Signing and Encryption (JOSE) header with the following elements:

- `kid` – The identifier of the AuthN Key Pair. See the figure below how to compute this value.
- `typ` – The media type of the complete JWT. For accessing AFIS B2B Services, the value must always be `jwt`.
- `alg` – The digital signature algorithm. For accessing AFIS B2B Services, the value must always be `RS256`. It corresponds to signing using a SHA-256 hash and RSA encryption.

How to compute the kid:

kid = UUID_V3(publicKey.encoded)

The kid is a UUID Version 3 and it is based on the public key material.
A UUID_V3 is obtained using a hashing so for every byte array input the output will always output the same UUID.
To compute the kid, the byte array to hash is the public key in its primary encoded format.

E.g.: Using Java, the kid can be computed in the following way:

```
import java.util.UUID;
import java.security.PublicKey;

PublicKey publicKey = getPublickJey();
UUID kid = UUID.nameUUIDFromBytes(publicKey.getEncoded());
```

The second part, commonly referred to as claims, has the following elements:

- `aud` – The Authorization Audience (see Prerequisites).

- `sub` – The subject of the JWT. It must be the client_id (see Prerequisites).
- `iss` – The issuer of the JWT. It must be the client_id (see Prerequisites).
- `exp` – The time when the JWT expires, expressed as an Epoch time (i.e. number of seconds elapsed since Thursday January 1st 1970 at 0:00:00 GMT/UTC).
- `iat` – The time when the JWT was issued, expressed as an Epoch time (i.e. number of seconds elapsed since Thursday January 1st 1970 at 0:00:00 GMT/UTC).
- `jti` – The unique identifier of the JWT. It is recommended to use a randomly non-sequential value, such as a Universally Unique Identifier (UUID).

As value for the **exp** claim we recommend to set it to a timestamp representing **1 minute** after it was created (iat). It should be a reasonable time window to complete any further computation happening from the moment the value is set and the moment the request is sent, including potential latency. Clock skew is taken into account on the authorization server. **In any case the gap between the iat and the exp value must not exceed 30 minutes!**

The signature of the JWT is a JSON Web Signature (JWS), as defined in RFC 7515, performed using the private key of the AuthN Key Pair.

1.2.3.2 Response

The figure below describes the response returned by the Authorization Server.

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8

{
    "access_token": "6DXJ2uof37yPvhCtoRO1luBOD1g",
    "scope": "read",
    "token_type": "Bearer",
    "expires_in": 59
}
```

The first line of the response contains the status code. It is followed by the headers above the blank line and the body under the blank line.

The body is a JSON document containing the following elements:

- `access_token` – The Access Token is opaque and therefore does not have any structure or expected content. It must be retrieved and stored in a character string.
- `scope` – The scope is expected to match the Resource Server Scopes and is provided for information.
- `token_type` – When accessing AFIS B2B Services, the value is always `Bearer`.
- `expires_in` – The number of seconds during which the Access Token remains valid.

1.2.4 Resource Server Exchange

The Resource Server Exchange is initiated by the Client that sends a request to the Resource Server in order to consume a service. It must include a valid Access Token obtained using the Authorization Server Exchange described above.

The Resource Server answers with a response containing business specific data or with an Unauthorized HTTP status code. For more details on these types of errors, check Resource Server Exchange.

It must be noted that an Access Token is only valid for a short duration and can only be used once. Every Resource Server Exchange therefore requires obtaining a new fresh Access Token, unless the AFIS B2B Service implements a mechanism to maintain trust, such as session or state.

1.2.4.1 Request

The figure below describes the request sent by the Client using HTTPS.

```
GET /protected-resource HTTP/1.1
host: resource-server.sample
accept: */*
Authorization: DPoP 6DXJ2uof37yPvhCtoRO1luBOD1g
DPop: eyJraWQiOiJkNTA...

...
```

The first line of the request contains the method that is dependent on the AFIS B2B Service invoked. It is followed by the headers above the blank line and the body under the blank line.

1.2.4.1.1 Headers

The mandatory request's headers are dictated by the AFIS B2B Service being invoked and are outside the scope of this document (including the headers in the example: host, accept.)

In addition to the headers specific to the AFIS B2B Service, the request must include:

- The **Authorization header**, containing the access token obtained using the Authorization Server Exchange (according to RFC6750).
  Attention: The authentication scheme must be "DPoP" thus notifying the Resource Server that the access token must be presented together with a DPoP Proof, as defined in the OAuth2 DPoP Draft Specification.
- The **DPoP header** containing a JSON Web Token encoded using Base64 further referred to as Resource Server **DPoP Proof**.

The Resource Server DPoP Proof must be signed with the private key of the DPoP Key Pair (see Prerequisites).

The figure below describes the Resource Server DPoP Proof.

```
{
  "typ": "dpop+jwt",
  "alg": "RS256",
  "jwk": {
    "kty": "RSA",
    "e": "AQAB",
    "use": "sig",
    "kid": "a34d9c68-7170-42d0-8792-7531c8a28237",
    "n": "lzweTKEPtiol--U..."
 }
}
{
 "htu": "https://resource-server.sample/protected-resource",
 "htm": "GET",
 "iat": 1615987167,
 "jti": "3476de44-5959-4510-8c51-93b8746e9569",
 "ath": "fUHyO2r2Z3DZ53EsNrWBb0xWXoaNy59IiKCAqksmQEo"
}
```

The Resource Server DPoP Proof is different from the Authorization Server DPoP proof. In particular `htu`, `htm`, `iat` and `jti` contain different values and `ath` is added.

The first part is a Javascript Object Signing and Encryption (JOSE) header with the following elements:

- `typ` – The media type of the complete JWT. For accessing AFIS B2B Services, the value must always be `dpop+jwt`.
- `alg` – The digital signature algorithm. For accessing AFIS B2B Services, the value must always be `RS256`. It corresponds to signing using a SHA-256 hash and RSA encryption.
- `jwk` – The public key of the DPoP Key Pair represented using the JSON Web Key (JWK) format.

The second part, commonly referred to as claims, has the following elements:

- `htu` – The URI of the Resource Server request endpoint.
- `htm` – The HTTP method used to invoke the Resource Server request endpoint.
- `iat` – The time when the JWT was issued, expressed as an Epoch time (i.e. number of seconds elapsed since Thursday January 1st 1970 at 0:00:00 GMT/UTC). When received by the Resource Server, the DPoP proof must not be older than a few minutes.
- `jti` – The unique identifier of the JWT. It is recommended to use a randomly non-sequential value, such as a Universally Unique Identifier (UUID).
- `ath` – The hash of the Access Token computed using SHA-256, encoded using Base64URL (with no padding).

How to compute the ath:

```
ath = base64URL( sha256_hash ( access_token ))
```

The signature of the JWT is a JSON Web Signature (JWS), as defined in RFC 7515, performed using the private key of the DPoP Key Pair.

1.2.4.1.2 Body

The content of the body is dictated by the AFIS B2B Service being invoked and is outside the scope of this document.

1.2.4.2 Response

When access is denied, the Resource Server should reply with the specific HTTP status code 401 (Unauthorized). This could happen if the request sent to the Resource Server does not contain an Access Token or if the Access Token is invalid, has expired or has already been used.

The figure below shows a response sent by a Resource Server when the Resource Server DPoP Proof is invalid. The `WWW-Authenticate` header provides more information on the root cause. It is optional and some Resource Servers might therefore not include it.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: DPoP realm="AfisRealm", error="invalid_token",
      error_description="Invalid DPoP key binding", algs="RS256"
```

In other situations, such as a successful invocation, the response sent by the Resource Server is dictated by the AFIS B2B Service being invoked and is outside the scope of this document.

2 Annexes

2.1 AFIS Security Best Practice for B2B

2.1.1 Abstract

This section describes security best practice for OAuth 2.0 adopted by AFIS. It is based on recommendations published in OAuth 2.0 Security Best Current Practice (SBCP) and explains which best practice have been implemented.

2.1.2 Recommendations

2.1.2.1 Client Authentication

From **2.5 Client Authentication of SBCP**:

Authorization servers SHOULD use client authentication if possible.

It is RECOMMENDED to use asymmetric (public-key based) methods for client authentication such as mTLS [RFC8705] or "private_key_jwt" [OpenID]. When asymmetric methods for client authentication are used, authorization servers do not need to store sensitive symmetric keys, making these methods more robust against a number of attacks.

The client authentication method implemented by AFIS is: **private_key_jwt**.

This method requires the Authorization Server to have access to the public key of the Client.

The Client needs to provide either the value of the public key or the JWKS_URI into the client registration info in the Authorization Server.

Resource Servers may use the same authentication method (**private_key_jwt**) when connecting to the introspect end-point of the Authorization Server.

2.1.2.2 Sender-Constrained Access Tokens

From **4.9.1.1.2. Sender-Constrained Access Tokens of SBCP**:

As the name suggests, sender-constrained access token scope the applicability of an access token to a certain sender. This sender is obliged to demonstrate knowledge of a certain secret as prerequisite for the acceptance of that token at a resource server.

A typical flow looks like this:

1. **The Authorization Server associates data with the Access Token that binds this particular token to a certain client**. The binding can utilize the client identity, but in most cases the Authorization Server utilizes key material (or data derived from the key material) known to the client.

2. This key material must be distributed somehow. Either the key material already exists before the Authorization Server creates the binding or the Authorization Server creates ephemeral keys.

AFIS uses key material. The Client application must be able to retrieve a key-pair (from a local or remote keystore) or must use ephemeral key-pairs.

AFIS distributes the public key on the application level, it sends the public key to the Authorization Server in the Token Request.

The Resource Server must also ensure that replay of the proof of possession is not possible.

AFIS uses the JTI claim contained in the received token for this purpose.

There exist several proposals to demonstrate the proof of possession in the scope of the OAuth working group (as mentioned SBCP), since none of these proposals has yet been approved, Authorization Servers available on the market do not allow implementing any of these proposals completely. The approach for AFIS is to implement a custom solution that is compatible with products on the market, such as OpenAM.

The solution put in place is based on the DPoP approach, one of the Sender-Constrained Access Tokens mechanisms presented in the SBCP, it mainly relies on the following specifications: DPOP, JWT [RFC7519] and Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) [RFC7800].

In practice it is DPoP with few customizations and exceptions to be compatible with ForgeRock Access Management (V 7).

In AFIS documentation we will refer to this approach as AFIS DPoP.

2.1.2.3 Audience Restricted Access Tokens

From **4.9.1.1.3. Audience Restricted Access Tokens**

Audience restriction essentially restricts access tokens to a particular resource server. The authorization server associates the access token with the particular resource server and the resource server SHOULD verify the intended audience. If the access token fails the intended audience validation, the resource server must refuse to serve the respective request.

AFIS implements audience restriction according to RFC 8707: Resource Indicators for OAuth 2.0.

2.2 AFIS DPoP

⚠ ***** Attention *****
The latest version of the AFIS B2B services still accept DPoP proof tokens, but they will not be actively validated any more. It is strongly advised to switch to OAuth2 authorization without DPoP.

This document is provided for legacy support. The most up-to-date technical specifications can be found in *AFIS B2B technical specifications with DPoP v3.0.pdf*

2.2.1 Introduction

This section is an extract, with few modifications, of the OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP) (Version 02). The main differences are highlighted in each paragraph and are listed here below for clarity:

- Only the flow for access token is described, Refresh Tokens are out of scope (since only Client Credential Flow is of interest at the moment and for this flow Refresh Tokens are not necessary).
- Only the flow for opaque access token is described, JWT Access Tokens are out of scope.
- To extend the flow to Authorization Servers that do not support (yet) the DPoP standard but that support RFC 7800 Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) 2 choices were made:
    1. Add an extra parameter, the "cnf_key" in the token request (according to the RFC 7800) to distribute the public key and bind the DPoP to the Access Token.
    2. In the implementation both token types ("DPoP" and "Bearer") are assumed to be bound to the DPoP Key Pair.
- JWK Confirmation Method in Token Introspection is used instead of “JWK Thumbprint”.
- The JWK signing algorithm supported is RS256 (RSA Signature with SHA-256)

The DPoP specification outlines a relatively simple application-level mechanism for sender-constraining OAuth Access Tokens. It enables a client to demonstrate proof-of-possession of a public/private key pair when presenting an Access Token to a Resource Server. The client distributes the public key of its key-pair to the Authorization Server when requesting an Access Token. Then the Authorization Server is able to bind issued tokens to the public key of the client's key pair.

It enables a Client to demonstrate proof-of-possession of a public/private key pair by including a "DPoP" header in an HTTP request. The value of the header is a JWT [RFC7519], referred to as DPoP proof, which enables the Authorization Server to bind issued tokens to the public part of the client's key pair.

Recipients of such tokens are then able to verify the binding of the token to the key pair that the client has demonstrated that it holds via signing the "DPoP" token, thereby providing some assurance that the client presenting the token also possesses the private key. In other

words, the legitimate presenter of the token is constrained to be the sender that holds and can prove possession of the private part of the key pair. DPoP can be used with public clients to sender-constrain access tokens and refresh tokens. With confidential clients, DPoP can be used in conjunction with any client authentication method to sender-constrain Access Tokens.

Opaque Access Tokens are tokens in a proprietary format that clients cannot access and typically contain some identifier to information in a server's persistent storage. To validate an opaque token, the recipient of the token needs to call the server (usually the /introspect end-point) that issued the token. Using opaque tokens instead of JWTs offer several advantages, such as:

- Signature is not needed;
- It is a meaningless string for everyone except for the Authorization Server;
- It does not require divulging anything about the chosen protocols;
- Since it needs to be introspected, the Authorization Server can provide information whether the Access Token has been revoked or not.

2.2.2 Main Objective

Under the attacker model defined in [OAuth 2.0 Security Best Current Practice (SBCP)] (SBCP), the mechanism defined by this specification aims to prevent token replay at a different endpoint. More precisely, if an adversary is able to get hold of an access token (or refresh token) because it set up a counterfeit Authorization Server or Resource Server, the adversary is not able to replay the respective token at another authorization or Resource Server.

2.2.3 Concept

The main data structure introduced by the DPoP specification is a DPoP proof JWT, described in detail below, which is sent as a header in an HTTP request. A Client uses a DPoP proof JWT to prove the possession of a private key corresponding to a certain public key. Roughly speaking, a DPoP proof is a signature over a timestamp and some data of the HTTP request to which it is attached.

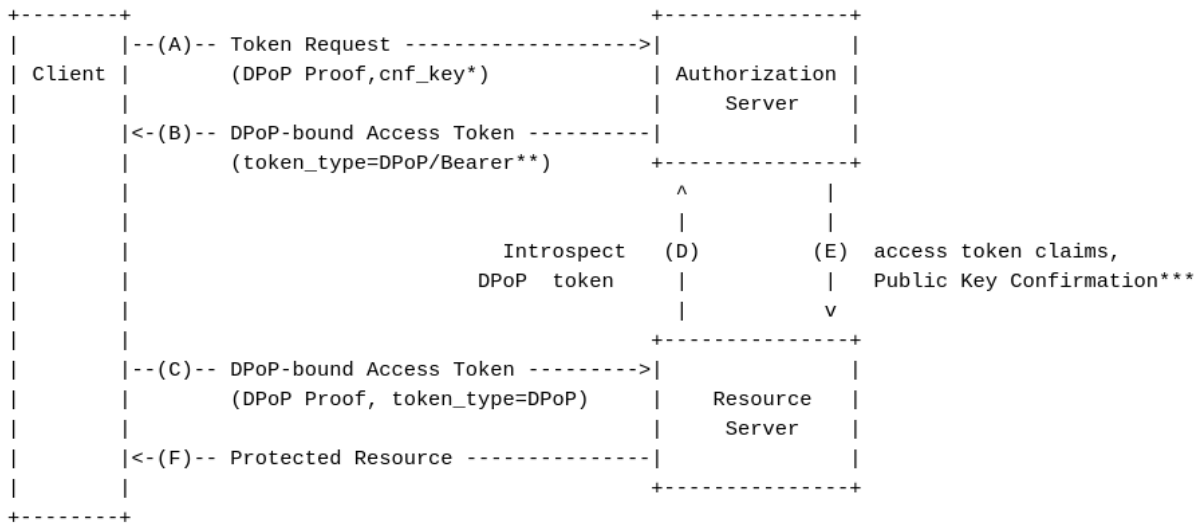


Figure 1: DPoP Flow

The basic steps of an OAuth flow with DPoP are shown in Figure 1:

- (A) In the Token Request, to obtain an Access Token (and potentially a Refresh Token), the Client attaches a DPoP proof to the request in the HTTP header.

*** If the Authorization Server does not support DPoP**

For Authorization Servers that do not support DPoP, but that support RFC 8700, the DPoP header added in the request to get the Access Token is ignored.
To allow the Authorization Server to bind the public key of the key pair used to create the DPoP to the Access Token the Client must provide it using the extra parameter "*cnf_key*" in the request body.
The value of the "cnf_key" parameter must be expressed as the JWK encoded using Base64.
For example:

```
ew0KICAgICJKV0siOiB7DQogICAgICAgICJhbGciOiAiUlMyNTYiLA0KICAgICAgICAiZSI6ICJ
BUUFCIiwNDQogICAgICAgICJraWQiOiAibXlQdWJsaWNKU09OV2ViS2V5Ig0KICAgIH0NCn0=
```

- (B) The Authorization Server binds (sender-constrains) the Access Token to the public key claimed by the client in the DPoP proof; that is, the Access Token cannot be used without proving possession of the respective private key.
- (C) To use the Access Token the Client has to prove possession of the private key by, again, adding a header to the request that carries the DPoP proof. The Resource Server needs to receive information about the public key to which the access token is bound. This information may be encoded directly into the Access Token (for JWT structured Access Token) or provided at the token introspection endpoint of the Authorization Server, as shown in the figure steps (D) and (E). The Resource Server verifies that the public key to which the Access Token is bound matches the public key of the DPoP proof.
- (F) The Resource Server refuses to serve the request if the signature check fails or the data in the DPoP proof is wrong, e.g., the request URI does not match the URI claim in the DPoP proof JWT.

The mechanism presented herein is not a client authentication method. In fact, a primary use case of DPoP is for public clients (e.g. single page applications) that do not use client authentication. Nonetheless, DPoP is designed such that it is compatible with "private_key_jwt" and all other client authentication methods. DPoP does not directly ensure message integrity but relies on the TLS layer for that purpose. See Section 9 of DPoP Spec. (Security Considerations) for details.

2.2.4 DPoP Proof JWT Syntax

DPoP introduces concept of a DPoP proof JWT, that is used for binding public keys and proving knowledge about private keys. The DPoP proof JWT is sent with an HTTP request using the "DPoP" header field.

2.2.4.1 DPoP Proof JWT Syntax

A DPoP proof is a JWT ([RFC7519]) that is signed (using JWS, [RFC7515]) using a private key chosen by the client (see below).

The header of a DPoP JWT contains at least the following parameters:

- "typ": type header, value "dpop+jwt" (REQUIRED).
- "alg": a digital signature algorithm identifier as per [RFC7518] (REQUIRED). MUST NOT be "none" or an identifier for a symmetric algorithm (MAC).
- "jwk": representing the public key chosen by the client, in JWK format, as defined in [RFC7515] (REQUIRED)

⚠ JWK signing algorithm

AFIS DPoP specification only supports RS256 (RSA Signature with SHA-256) at the moment. In the future, other asymmetric algorithms might be supported.

The body of a DPoP proof contains at least the following claims:

- "jti": Unique identifier for the DPoP proof JWT (REQUIRED). The value MUST be assigned such that there is a negligible probability that the same value will be assigned to any other DPoP proof used in the same context during the time window of validity. Such uniqueness can be accomplished by encoding (base64url or any other suitable encoding) at least 96 bits of pseudorandom data or by using a version 4 UUID string according to [RFC4122]. The "jti" SHOULD be used by the server for replay detection and prevention, see Section 9.1 of DPoP Spec..
- "htm": The HTTP method for the request to which the JWT is attached, as defined in [RFC7231] (REQUIRED).
- "htu": The HTTP URI used for the request, without query and fragment parts (REQUIRED).
- "iat": Time at which the JWT was created (REQUIRED).

Figure 2 shows the JSON header and payload of a DPoP proof JWT.

```
{
  "typ":"dpop+jwt",
  "alg":"RS256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}.{
  "jti":"-BwC3ESc6acc2lTc",
  "htm":"POST",
  "htu":"https://example.authorization-server.com/token",
  "iat":1562262616
}
```

Figure 2: Example JWT content for "DPoP" proof header

Note: To keep DPoP simple to implement, only the HTTP method and URI are signed in DPoP proof tokens. The idea is sign just enough of the HTTP data to provide reasonable proof-of-possession with respect to the HTTP request. But that it be a minimal subset of the HTTP data so as to avoid the substantial difficulties inherent in attempting to normalize

HTTP messages. Nonetheless, DPoP proof tokens can be extended to contain other information of the HTTP request (see also Section 9.4 of DPoP Spec).

2.2.4.2 Checking DPoP Proofs

To check if a string that was received as part of an HTTP Request is a valid DPoP proof, the receiving server MUST ensure that

1. the string value is a well-formed JWT,
2. all required claims are contained in the JWT,
3. the "typ" field in the header has the value "dpop+jwt",
4. the algorithm in the header of the JWT indicates an asymmetric digital signature algorithm, is not "none", is supported by the application, and is deemed secure,
5. that the JWT is signed using the public key contained in the "jwk" header of the JWT,
6. the "htm" claim matches the HTTP method value of the HTTP request in which the JWT was received (case-insensitive),
7. the "htu" claims matches the HTTP URI value for the HTTP request in which the JWT was received, ignoring any query and fragment parts,
8. the token was issued within an acceptable timeframe (see Section 9.1 of DPoP Spec), and
9. that, within a reasonable consideration of accuracy and resource utilization, a JWT with the same "jti" value has not been received previously (see Section 9.1 of DPoP Spec).
10. If presented with an access token to a protected resource, the server MUST ensure that the value of the "ath" claim equals the hash of the access token that has been presented along side the DPoP proof.

Resource Server implementation details!
Servers SHOULD employ Syntax-Based Normalization and Scheme-Based Normalization in accordance with Section 6.2.2 and Section 6.2.3 of [RFC3986] before comparing the "htu" claim.

2.2.5 Token Request (Binding Tokens to a Public Key)

To bind a token to a public key in the token request, the client MUST provide a valid DPoP proof JWT in a "DPoP" header. The HTTPS request shown in Figure 3 illustrates the protocol for this (with extra line breaks for display purposes only).

```
POST /token HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Authorization: Basic ZXhhbXBsZS1jbGllbnQ6ZXhhbXBsZS1jbGllbnQtc2VjcmV0
DPoP: eyJraWQiOiJkODU...
Host: authorization-server.com

grant_type=client_credentials
&cnf_key=eyJqd2siOiB7Imt....
&scope=read
```

Figure 3: Token Request for a DPoP sender constrained token

The "DPoP" HTTP header MUST contain a valid DPoP proof JWT. If the DPoP proof is invalid, the Authorization Server issues an error response per Section 5.2 of [RFC6749] with "invalid_dpop_proof" as the value of the "error" parameter.

To sender-constrain the Access Token, after checking the validity of the DPoP proof, the Authorization Server associates the issued access token with the public key from the DPoP proof (or the ***cnf_key*** parameter if the Authorization Servers do not support DPoP), which can be accomplished as described in Section 6 of DPoP Spec. A "token_type" of "DPoP" in the Access Token response signals to the client that the access token was bound to its DPoP key and can used as described in Section 7.1 of DPoP Spec.

The example response shown in Figure 4 illustrates such a response.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token": "Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU",
 "token_type": "DPoP",
 "expires_in": 2677,
}
```

Figure 4: Access Token Response

⚠ **** If the Authorization Server does not support DPoP**

If the Authorization Servers does not support DPoP, the "token_type" will be "Bearer" instead of "DPoP", as in the example below.

```
{
  "access_token": "Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU",
  "token_type": "Bearer",
  "expires_in": 2677,
}
```

A client system that wants to communicate with an AFIS system does not need to rely on the token type to assume that the Access Token was bound to its DPoP key.

The Access Token issued by the Authorization Server to Clients using the AFIS DPoP specification are always bound to the DPoP key, even if of type bearer.

2.2.6 Public Key Confirmation

Resource Servers MUST be able to reliably identify whether an Access Token is bound using DPoP and ascertain sufficient information about the public key to which the token is bound in order to verify the binding with respect to the presented DPoP proof (see Section 7.1 of DPoP Spec). Such a binding is accomplished by associating the public key with the token in a way that can be accessed by the protected resource, such as embedding the JWK or the JKT (an hash of the JWK) in the issued access token directly, using the syntax described in Section 6.1 of DPoP Spec, or through token introspection as described in Section 6.2 of DPoP Spec.

Resource Servers supporting DPoP MUST ensure that the public key from the DPoP proof matches the pubic key to which the Access Token is bound.

***** Public Key Confirmation**

As stated in Section 6. of DPoP Spec, other methods of associating a public key with an access token are possible, per agreement between the Authorization Server and the protected resource.
In the context of AFIS, the syntax described in the following paragraph is used (JWK Confirmation Method in Token Introspection). It consists in returning the JWK in the cnf member of the response of the token Introspection instead of the JKT (that is a hash of the JWK) as proposed in the Section 6. of DPoP Spec. Moreover, the use of the JKT is still under discussion in the IETF work group of the DPoP standard.

2.2.6.1 JWK Confirmation Method in Token Introspection

OAuth 2.0 Token Introspection [RFC7662] defines a method for a protected resource to query an Authorization Server about the active state of an Access Token as well as to determine metainformation about the token.

For a DPoP-bound Access Token, the hash of the public key to which the token is bound is conveyed to the protected resource as metainformation in a token introspection response. The hash is conveyed using the same "cnf" content with "jkt" member structure as the JWK thumbprint confirmation method, described in Section 6.1 of DPoP Spec, as a top-level member of the introspection response JSON. Note that the Resource Server does not send a DPoP proof with the introspection request and the Authorization Server does not validate an access token's DPoP binding at the introspection endpoint. Rather the Resource Server uses the data of the introspection response to validate the access token binding itself locally. The example introspection request in Figure 9 and corresponding response in Figure 6 illustrate an introspection exchange for the example DPoP-bound access token that was issued in Figure 4.

```
POST /introspect HTTP/1.1
Host: example.authorization-server.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic ZXhhbXBsZS1jbGllbnQ6ZXhhbXBsZS1jbGllbnQtc2VjcmV0

token=Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU

                Figure 5: Example Introspection Request
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
  "active": true,
  "sub": "someone@example.authorization-server.com",
  "iss": "https://example.authorization-server.com",
  "nbf": 1562262611,
  "exp": 1562266216,
    "cnf": {
   "jwk": {
   "kty": "RSA",
   "e": "AQAB",
   "use": "sig",
```

```
      "kid": "d857a8f4-1cee-496b-b898-d9f36279bf16",
      "n": "lzweTKEPtiol--
U1MwUfrsIaIsm697wXnDawk9_c_KQ_rtQ4iiMg9RiNJoA4Cm6K96Ueh8jy1xBUBf_..."
            }
     }
   }
```

Figure 5: Example Introspection Response for a DPoP Bound Access Token

For further details about the following subjects, the reader is invited to consult the original documentation of the DPoP specification:

Protected Resource Access

Security Considerations

IANA Considerations

2.2.7 AFIS DPoP Flow Implementation

2.2.7.1 Overview

The AFIS DPoP flow described here is implemented using the standard OAuth2 Client Credential Flow (as per OAuth2 Client Credentials Grant) and is using ***private_key_jwt*** as authentication method (as per Authentication Method by means of the private_key_jwt).

For the Authorization Server to be able to validate the signature of the JWT in the **Client Assertion**, the public key of this key pair needs to be in the client configuration of the Authorization Server for each client. Regarding this public key, there are two options: a static **JWK** or a **JWKS_URI**.

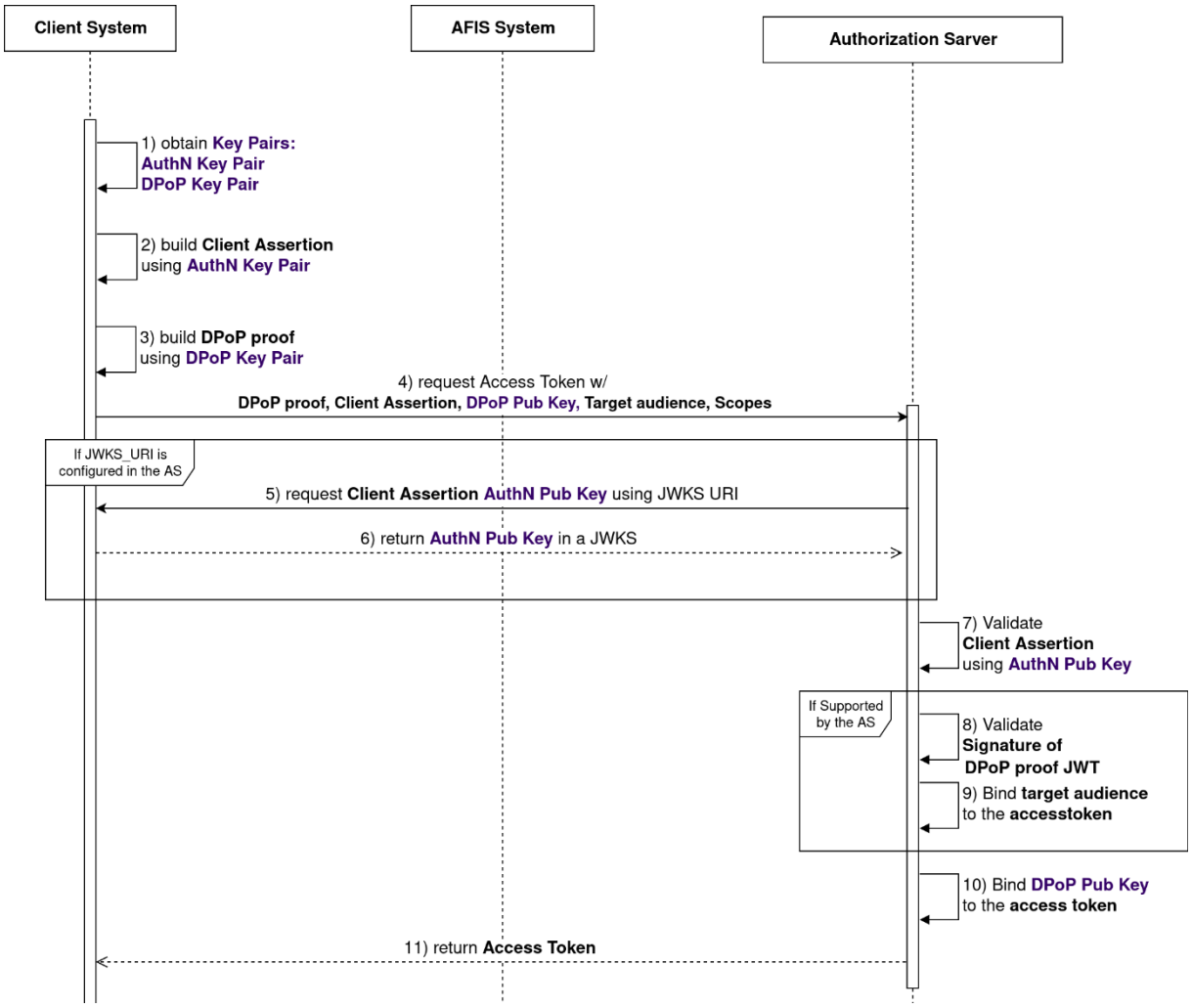
The **static JWK** is the public key in JWK format configured directly on the client configuration of the Authorization Server, e.g. set through a back-office form.

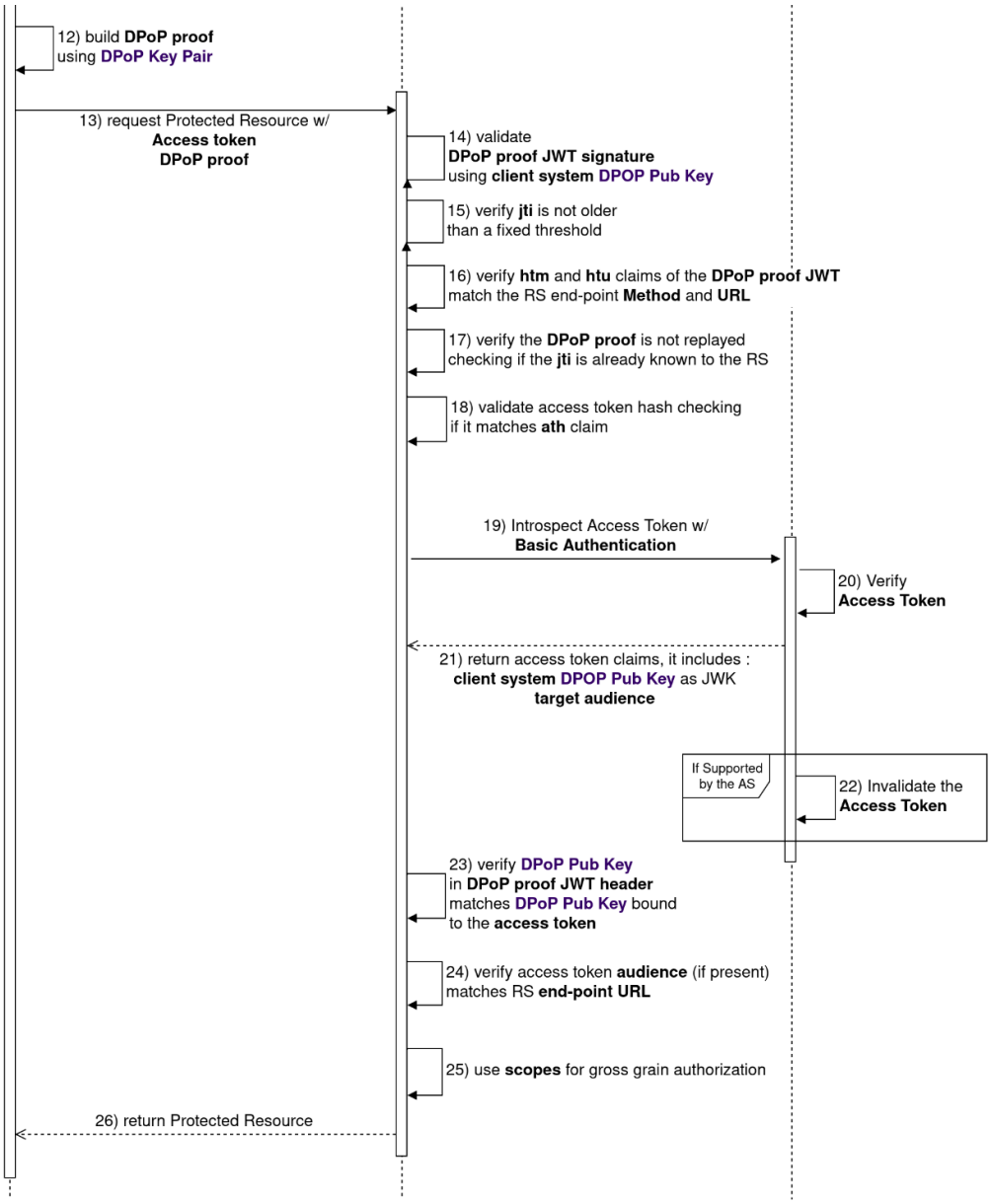
The **JWKS_URI** is the URI of an end-point where its public key is returned in a JWK set. The client must expose this end-point and return a JSON WEB KEY SET (see JSON Web Key (JWK) specification). After receiving the access token request with the parameter client assertion, the Authorization Server invokes this endpoint to obtain the set of public keys (JWKS). Since the client assertion contains the identification of the key pair which was used to sign it, the Authorization Server is able to obtain the matching public key from the JWK set and validate the signature.

The type of Access Token issued by the Authorization Server is an ***opaque token***, the Resource Server (AFIS System in the schema presented) will need to use the introspect end-point of the Authorization Server to get the details, as claims, of the ***access token***.
The Resource Server also use ***private_key_jwt*** as authentication method (as per Authentication Method by means of the private_key_jwt) to call the introspect end-point.
The Resource Server implements ***private_key_jwt*** as authentication method as described for the client system.

2.2.7.2 Flow Sequence Diagram

The diagram continues in the next page





2.2.7.3 Flow Sequence Diagram steps

1. The client system needs to obtain 2 key pairs to implement the DPoP flow: one for authentication to the Authorization Server (**AuthCS Key Pair**) and one for DPoP proof signature (**DPoP Key Pair**).
   The same key pair can be used for both purposes but it is advised to use a stable key pair for authentication and ephemeral (rotating) key pair for DPoP proof signature.
2. The client system creates a client assertion according to Authentication Method by means of the private_key_jwt using the **AuthCS Key Pair**.
3. The client system creates a DPoP proof according to AFIS DPoP using the **DPoP Key Pair**.
4. The request to the Authorization Server to get the Access Token must contain the following header:

a. **DPoP**: The DPoP proof built in step 3 according to AFIS DPoP.
   And the following request parameters:
b. **grant_type**: The Oauth2 flow to use (for machine-to-machine use case must be "client_credentials")
c. **client_assertion_type**: The Client Assertion Type (urn:ietf:params:oauth:grant-type:jwt-bearer) as specified in Authentication Method by means of the private_key_jwt.
d. **client_assertion**: The Client Assertion built in step 2 for the authentication according to Authentication Method by means of the private_key_jwt.
e. **cnf_key**: The public key of the **DPoP Key Pair** used in step 3 for building the DPoP header.
f. **resource**: The Target Audience of the access token requested according to RFC 8707: Resource Indicators for OAuth 2.0.
g. (Optional) **scopes**: the needed Scopes to restrict the use of the access token only to the scopes specified.

Access Token Request

```
POST /token HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
DPoP: eyJraWQiOiJkODU...
Host: example.authorization-server.com

grant_type=client_credentials
&client_id=example-client
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-
type%3Ajwt-bearer
&client_assertion=eyJraWQiOiIyNzF...
&cnf_key=eyJqd2siOiB7Imt....
&resource=https%3A%2F%2Fexample.resource-server.com2Fprotected-resource
&scope=read
```

5. If the client system configuration in the Authorization Server is set to JWKS_URI as Public Key selector, the Authorization Server requests the Public Key of the **AuthN Key Pair** to the client system in order to validate the signature of the Client Assertion used for authentication
6. The client system returns a JWKS (JSON WEB KEY SET) containing the Public Key of the **AuthN Key Pair** as JWK.
   If the Public Key selector is set to JWKs then steps 5 and 6 will be skipped.

Client JWKS_URI Response

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "n":"rkESKI_CEu2NLlMFCcFxyADWcwrr_8zn0BwZHaeJ....",
      "kid": "271e3768-f92c-470a-a108-fdb7a3e0535a"
    }
  ]
}
```

7. The Authorization Server validates the signature of the client assertion JWT using the Public Key of the **AuthN Key Pair** obtained at point 6.
8. If the Authorization Server supports the DPoP specification, it validates the signature of the DPoP proof using the public key of the **DPoP Key Pair**.
9. If the Authorization Server supports the RFC 8707: Resource Indicators for OAuth 2.0 it will bind the target audience to the access token.
10. The Authorization Server binds the public key of the **DPoP Key Pair** to the Access Token.
    If the Authorization Server supports the DPoP specification it gets the public key from the header of the DPoP proof sent in step 4 (point a), if not it gets it from the param "**cnf_key**" of step 4 (point e)
11. The Authorization Server returns the access token as an opaque token of type "Bearer" according to AFIS DPoP.

Access Token Response

```
{
   "access_token":"b6jTq0d6L9lGLoRPlyeiQQnS1L0",
   "scope":"read",
   "token_type":"Bearer",
   "expires_in":3599
}
```

12. The client system builds a fresh new DPoP proof using the **DPoP Key Pair** (the same key pair used for the first DPoP Proof used to get the access token since they must share the same public key).
    This new DPoP proof will be used in the request to the protected resource.
13. The client system requests the protected resource to the Resource Server API end-point according to AFIS DPoP. The request must contain:
    a. **Authorization Header:** The access token retrieved in step 11.
    b. **DPoP Header**: The DPoP proof built in step 12.

API Request w/ DPoP proof and Access Token

```
GET /protected-resource HTTP/1.1
host: https://example.resource-server.com
accept: */*
Authorization: DPoP b6jTq0d6L9lGLoRPlyeiQQnS1L0
DPoP: eyJraWQiOiJkODU...
```

14. The Resource Server validates the DPoP proof signature (using the public key of the **DPoP Key Pair** present in the JWT header of the DPoP proof) to verify the integrity of the DPoP proof payload.
15. The Resource Server verifies that the "**iat**" claim of the DPoP proof is not older than a fixed threshold, otherwise will discard the DPoP proof.
16. The Resource Server verifies that the "**htm**" and "**htu**" claims contained in the DPoP proof matches respectively the HTTP Method and the URL of the API end-point of the protected resource according to AFIS DPoP.
17. The Resource Server verifies that the "**jti**" has not been seen before and decline the HTTP request otherwise.

18. The Resource Server verifies that the "**ath**" contains the hash of the access token sent in the HTTP Authorization header of the request.
19. The RS introspects the Access Token according to RFC 7662: OAuth 2.0 Token Introspection. It use basic authentication as authentication method towards the AS.
20. The AS verifies that the access token is still valid.
21. The Authorization Server returns the claims associated to the Access Token according to RFC 7662: OAuth 2.0 Token Introspection. In the JSON response the following members must be present:
    a. **"cnf"** : This member contains the public key of the **DPoP Key Pair** used by the client system to sign the DPoP proof. The public key is returned as a **JWK** according to the RFC 7800 Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)
    b. (Optional, If supported by the Authorization Server) **"aud"** : The target audience specified at step 4 in the "resource" param according to RFC 8707: Resource Indicators for OAuth 2.0.

Introspect Access Token Response

```
{
  "active": true,
  "scope": "read",
  "realm": "/afis-realm",
  "client_id": "example-client",
  "user_id": "example-client",
  "token_type": "Bearer",
  "exp": 1616003021,
  "sub": "example-client",
  "iss": "https://example.authorization-server.com/afis-realm",
  "cnf": {
    "jwk": {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "d857a8f4-1cee-496b-b898-d9f36279bf16",
      "n": "lzweTKEPtiol--U1MwUfrsIaIsm697wXn..."
    }
  },
  "authGrantId": "EHzRZoyGaXZN9YVWjljTmrmWJbo",
  "auditTrackingId": "0cbde954-a16c-42d1-9802-c567adec7c13-561821"
}
```

22. The Authorization Server may validate (if it supports this feature) so it can only be used once.
23. The Resource Server verifies that the public key (**DPoP PK**) of the **DPoP Key Pair** present in the JWT header of the DPoP proof matches the public key (**DPoP Pub Key**) bound to the access token (retrieved at step 21 as claim "jwk") as specified in AFIS DPoP.
24. The Resource Server verifies that the audience bound to Access Token (if present in the "aud" claim returned by the introspect end-point) matches the URL of the API end-point of the protected resource according to RFC 8707: Resource Indicators for OAuth 2.0.

25. The Resource Server can use the scopes (if any scope is present in the "scopes" claim returned by the introspect end-point) for fine grain authorization to the appropriate resources.
26. The Resource Server can return the Protected Resource if all the validation are successful.